Disaggregating the BNG in SEBA

Table of contents

Table of contents **1**

Prerequisites **2**

Introduction **2**

Scope of this document 3

Acknowledgments 3

Target architecture **4**

Packet headers 5

Data packets 5

Control packets

BNG-c **6**

Implementation options 6

Native BNG-c 7

Relay BNG-c 7

External BNG-c 8

BNG-u runtime control API 8

Integration with Trellis 9

P4 pipeline design **10**

Fabric.p4 pipeline recap 11

Double VLAN handling in fabric.p4 12

BNG-u integration with fabric.p4 12

Design goals 12

Initial P4 implementation and PTF tests 13

Overview 13

Upstream ingress 15

Downstream ingress 16

Upstream egress 17

Downstream egress 18

Missing features 18

IPoE-based service delivery 18

Wholesale tunnel relay (e.g. via L2TP) 18

Punt rate limiting 18

MTU and fragmentation 19
Lawful intercept 19
Support for L2 and L3 VNF chains on the compute nodes 19

Other BNG implementations **19**

Prerequisites

This document assumes familiarity with:

- SEBA reference design
- ONOS architecture
- Trellis and support for P4-capable switches (fabric.p4)
- P4 language concepts

Introduction

The goal of this document is to provide a design for the disaggregation of the Broadband Network Gateway (BNG) functionality in SEBA.

Today, the SEBA exemplar platform assumes an external BNG, placed between the aggregation switch (AGG SW) and the operator core network. We propose to:

1. Disaggregate the BNG in two pieces, user plane (BNG-u) and control plane (BNG-c);
2. Implement the BNG-u by exploiting merchant silicon in the existing data plane of SEBA, distributing its functions between the AGG SW and the OLT;
3. Implementing BNG-c as an application running on top of ONOS.

At high level, the BNG-u is expected to provide packet processing functions such as:

- Aggregation and termination of double-vlan tagged traffic, and routing
- Subscriber tunnel termination (e.g. PPPoE)
- Wholesale tunnel relay (e.g. L2TP)
- Hierarchical QoS for downstream traffic (H-QoS)
- Accounting
- Anti-spoofing
- Lawful interception (per-subscriber wiretap)

BNG-c is responsible of handling:

- Authentication and authorization of new subscriber connections, for example by acting as a proxy to other AAA systems/apps;
- Changes to forwarding state of BNG-u with subscriber-specific information (session identifiers, QoS parameters, etc.)
- Interactions with systems outside the SEBA pod, including northbound routers.

For the BNG-u, the required packet processing functions can be distributed in two places:

1. **ASG SW**: This is the same as the AGG SW, but since it now includes service edge capabilities, it is called Aggregation and Service Gateway (ASG). We plan to use P4 and programmable ASICs like Barefoot Tofino to describe and implement functions such as tunnel termination, accounting (i.e. counters) and some downstream QoS such as policing.
2. **OLT**: We plan to use the Broadcom Qumran chip inside the EdgeCore white box OLT (ASFvOLT16), or in the case of OLTs that do not include aggregation, in a separate Qumran-based aggregation switch. These will implement the rest of the QoS functionalities for the downstream, such as traffic prioritization using 3 levels of hierarchical scheduling, e.g., with queues and schedulers per PON, per subscriber, and per class of traffic. QoS for the upstream direction is optionally performed on the Residential Gateway (RG) or Optical Network Unit (ONU), and then policed by the OLT or access aggregation switch

Scope of this document

This document focuses on the BNG-u capabilities required in the ASG SW such as:

- Termination of L2 and subscriber tunnels using double VLAN tags (QinQ) and PPPoE headers
- Anti-spoofing
- Accounting, i.e. per subscriber traffic counters
- Preliminary downstream QoS (traffic policing)

Important: The requirements and design for the downstream QoS functionalities concerning prioritization and executed on the OLT are yet to be determined and will NOT be discussed here.

Some other functionalities that are commonly found in other BNG implementations are described in Section Missing features and will be tackled at a later stage.

Moreover, we consider only the case where the access network uses IPv4, however, the same considerations can be easily applied to IPv6 as well.

Finally, we discuss implementation options of the BNG-c component, but we do NOT discuss its functional requirements.

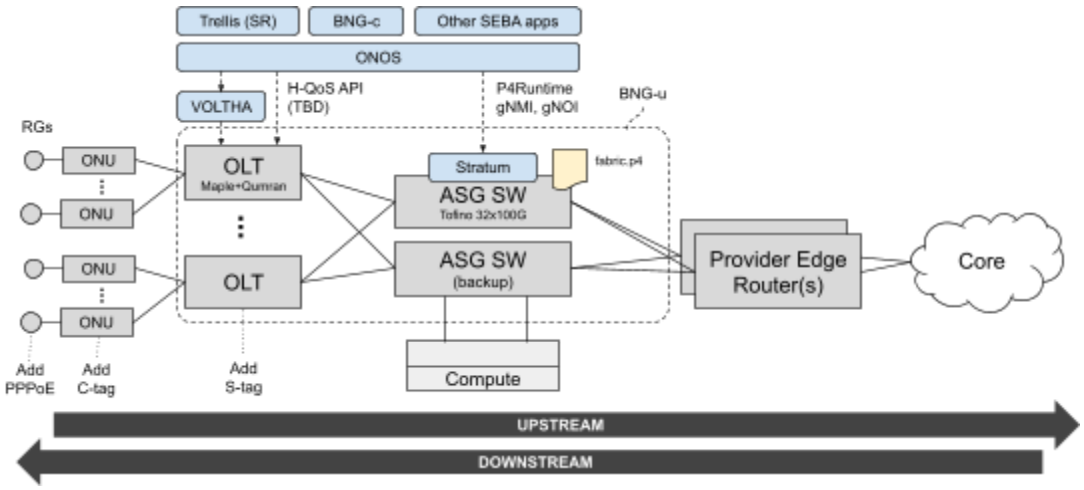
Acknowledgments

The design presented here is based on a contribution from Deutsche Telekom to ONF for a P4-based implementation of a Service Edge that includes BNG functionalities: https://github.com/opencord/p4se

We apply key learnings from DT's contribution to the SEBA case and re-use some of their P4 code.

Target architecture

The following diagram depicts the SEBA exemplar architecture with disaggregated BNG functionality, where the BNG-u is distributed between the ASG SW and the OLT. Following the SEBA terminology, the AGG SW, since it now provides "service edge" capabilities, is called Aggregation and Service Gateway (ASG).



ASG is a P4 programmable switch, programmed using a version of [fabric.p4] extended with BNG capabilities. Fabric.p4 is an existing P4 program currently distributed as part of ONOS, it was created to provide support for Trellis on P4-programmable switches. The pipeline defined by fabric.p4 provides standard L2-L3 functionalities as required by Trellis. We describe here the extension required to fabric.p4 to provide some of the BNG-u capabilities. The remaining BNG-u functionality such as hierarchical QoS (H-QoS) is provided as part of the OLT.

The ASG connects to one or more Provider Edge (PE) routers, which in turn provide access to the core network. For redundancy, multiple ASG SWs can be used, however, the current design does not consider a full fabric with spines, which might prevent scaling the number of compute nodes, OLTs, or routers.

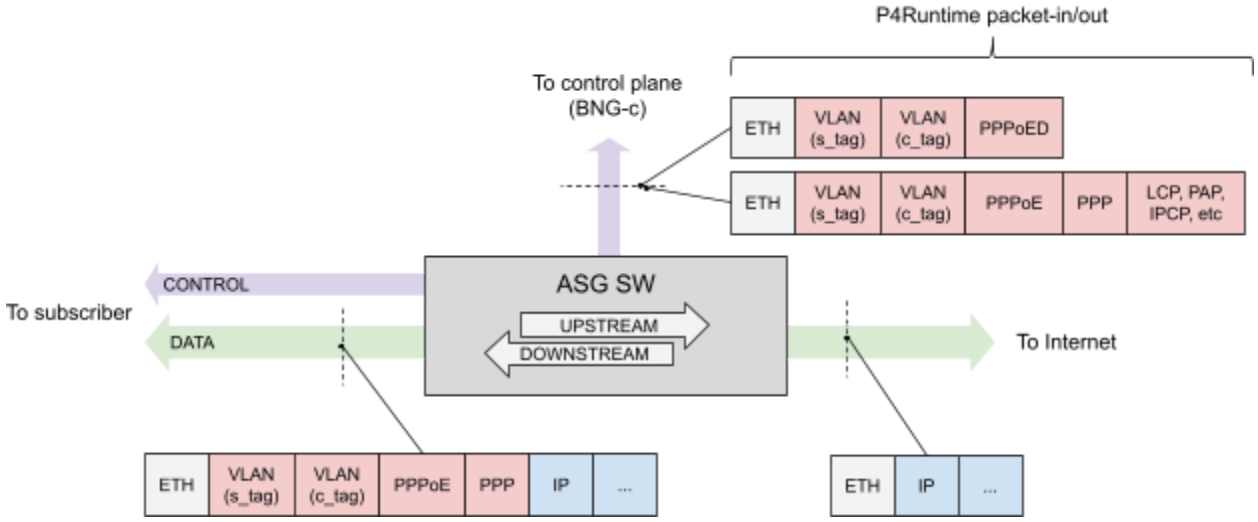
The ASG runs Stratum and is controlled by ONOS using protocols such as P4Runtime, gNMI, and gNOI. The OLT is controlled by Voltha as it is in SEBA today, however, we envision an additional API that would allow the control of the H-QoS functionality for the BNG-u. The definition of such API is outside the scope of this document.

ONOS runs three groups of apps:

1. **Trellis apps**: such as SegmentRouting (SR), etc., used to provide basic L2/L3 forwarding as part of the fabric between the OLT and the PE router. The same Trellis apps used today with OFDPA-based switches can be used with Stratum-based ones and fabric.p4, as a “pipeliner” driver in ONOS provides a mapping for the forwarding configuration (FlowObjective) generated by the Trellis apps, to the specific tables of fabric.p4.
2. **BNG-c**: this is BNG control plane app, responsible of processing control plane packets coming from the RG to authenticate and configure PPPoE sessions. PPPoE control plane packets are intercepted by the BNG-c app by means of P4Runtime packet-in (similar to OpenFlow packet-in).
3. **SEBA apps:** such as vOLT, Sadis, AAA, dhcpl2relay, etc.

Packet headers

The data plane should be able to handle packet headers as depicted in the following figure:



Data packets

Upstream data packets coming from the OLTs are expected to be encapsulated in 2 VLAN tags (802.1ad) and a PPPoE header. The VLAN tags are used to:

- The **outer** one identifies the PON network/OLT, and it’s usually called **S-tag**
- The **inner** one identifies the ONU, and it’s usually called **C-tag**

The combination of S-tag and C-tag is used to uniquely identify a L2 access line from the BNG to the ONU. The PPPoE header carries a *session_id*. The combination of S-tag, C-tag and session_id uniquely identifies a subscriber.

For the upstream direction, the ASG SW is responsible of removing the VLAN tags and PPPoE header before routing packets to the PE router.

Analogously, IP packets in the downstream direction are expected to be encapsulated in the same VLAN and PPPoE headers before being routed to the subscriber RG.

Control packets

Control packets are used for session establishment by the subscriber, including authentication. These are expected to use PPPoE Discovery (PPPoED) or PPP protocol packets such a LCP, CHAP, etc. used for authentication and session configuration. An example of PCAP trace including these control packet is available here.

The BNG-u is responsible of punting these packets to the BNG-c by means of P4Runtime packet-ins without further modifications. Similarly, the BNG-u is responsible of delivering replies from the BNG-c as P4Runtime packet-out.

BNG-c

The BNG control plane is expected to perform functions like authentication and configuration of new subscriber connections. The implementation of these functions depends on the protocol used for service delivery. In the case of PPPoE-based service delivery, the control plane is expected to:

- Handle PPPoED packets for session discovery
- Handle PPP LCP packets for link control
- Handle PPP IPCP packets for address assignment
- Handle authentication via PPP protocols such as PAP, CHAP, EAP.
- And more.

As a consequence of the handling of these packets, the BNG-c is also responsible for updating the state of the BNG-u to terminate and route the PPPoE sessions (i.e. data packets exchanged between the subscriber and the Internet), as well as configuring the QoS parameters.

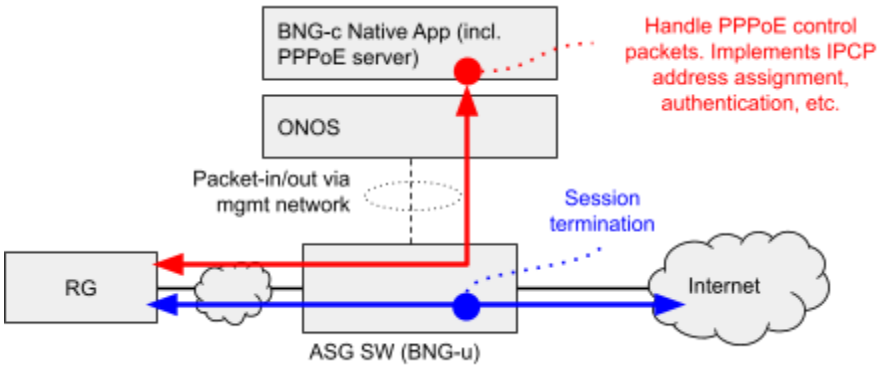
In the following, we describe different options for BNG-c implementation, as well as integration with ONOS to update the BNG-u state and interaction with the Trellis apps to provide routing functions.

Implementation options

The SEBA exemplar platform should allow for different ways of implementing the BNG control plane, including the integration with existing PPPoE servers already used by operators to provide functions like IPCP-based address assignment, authentication, etc. We describe here three options, namelly *native*, *relay* and *external*. These options are equivalent and the overall

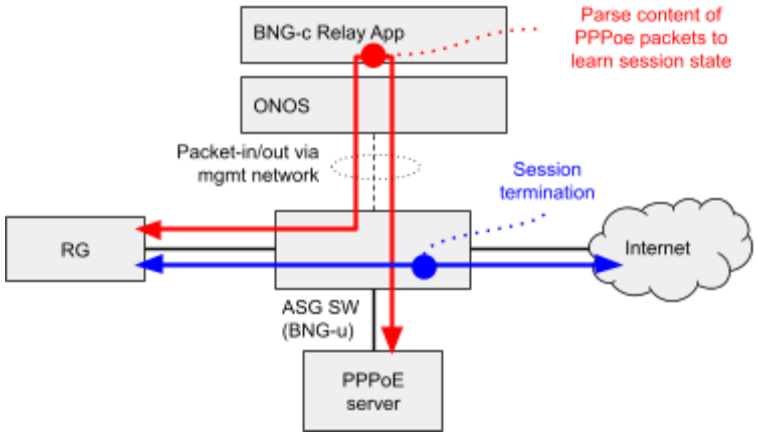
SEBA exemplar design does not prescribe any of them. Which one to use should be left to the operator's choice.

Native BNG-c



This is the case where the BNG-c is implemented entirely in Java as a native application running on top of ONOS. PPP control packets are intercepted from the ASG SW and delivered to the app by means of P4Runtime packet-ins via the pod management network. The app implements logic such as IPCP address assignment and authentication, generating replies to such control packets and managing BNG-u forwarding state. Replies to control plackets are delivered to the RG by means of P4Runtime packet-out. The native option provides the best integration with ONOS as it allows access to the entire network state, however, it requires to implement from scratch all PPPoE server functions.

Relay BNG-c



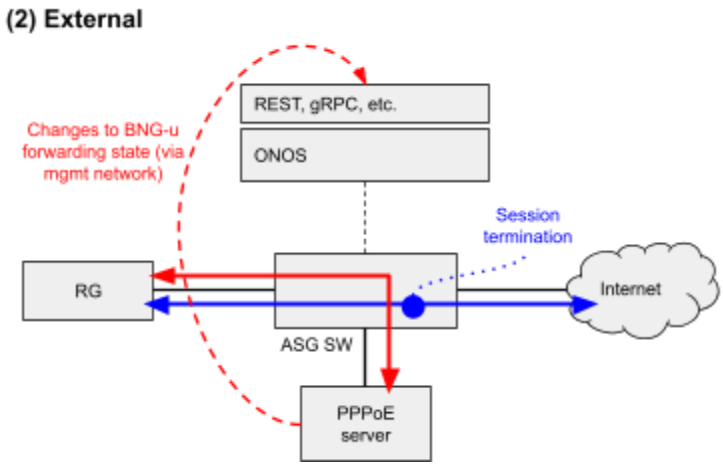
This case is similar to the native one but allows the integration with an existing PPPoE server. An app running on top of ONOS acts as a relay to an external server attached to the data plane network. PPPoE control packets are intercepted by means of packet-ins and relayed to the server via packet-out. Replies from the server are also relayed to the subscriber using the same mechanism. While relaying packets, the app parses the content of the PPPoE control packets to learn about subscriber sessions and to modify the BNG-u forwarding state. For example, by

looking at the content of PPPoED packets (PADI, PADO, PADS, PADS) the app can learn the PPPoE session ID associated with an RG. Similarly, by looking at the IPCP packets, the app can learn the IP address assigned to the RG.

This option allows re-using an existing, potentially unmodified implementation of a PPPoE server already deployed in an operator network. However, it's important to note how the PPPoE control packets are relayed through the ASG SW management port, usually 1G, and through the pod management network. This might pose limitations to the rate of PPPoE control packets handled inside the same pod.

External BNG-c

To avoid using the pod management network to transport PPPoE control packets, we can consider the case of an external PPPoE server where control packets are forwarded entirely in the data plane network. However, to manage the BNG-u forwarding state we need to interface the external PPPoE server with ONOS. A potential option for such integration is that where ONOS provides northbound APIs (e.g. using REST or gRPC) that can be used by the PPPoE server to configure session termination on the BNG-u. The downside of this option is that operators will need to modify existing servers to support this new API.



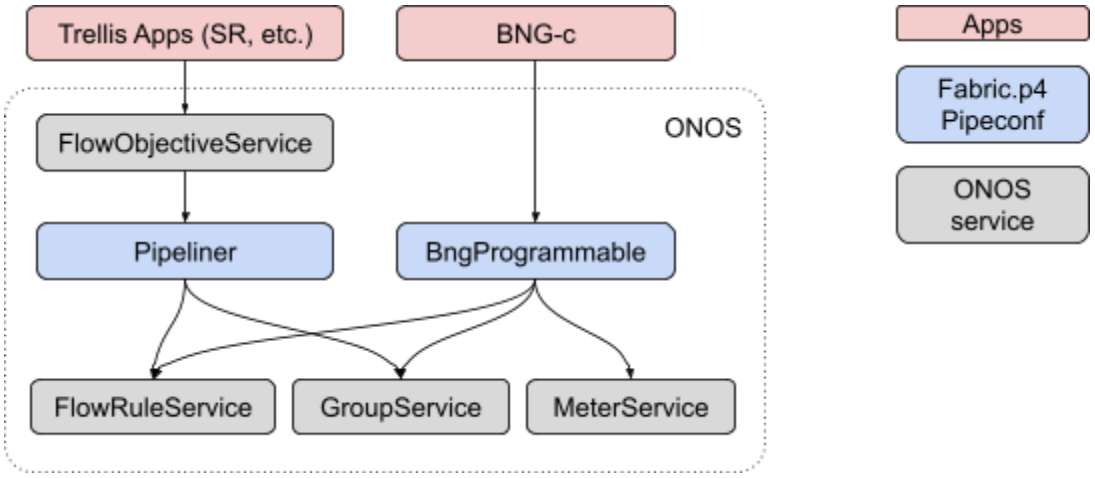
BNG-u runtime control API

To facilitate the operations of the latter and to allow independent evolution of the BNG-u implementation (including the P4 pipeline design), we propose to implement an high-level API in ONOS to modify forwarding state of the BNG-u. Such API, would expose methods convenient to the BNG-c to manage the termination of subscriber sessions, QoS configuration, and to read counters.

We propose such API to be defined as an ONOS driver behaviour named `BngProgrammable`, which implementation would be provided as part of the fabric.p4 pipeconf. The BngProgrammable would serve a purpose similar to the FlowObjective API for the Trellis apps, offering a pipeline-independent way of manipulating forwarding state, internally mapped to pipeline-specific state such as FlowRule, Groups, etc.

The BngProgrammable API is yet to be defined, however we expect it to offer methods to:

- Discover and enable attachments, e.g., set up subscriber lines (line_id, pppoe_session_id)
- Authenticate requests for attachments, e.g., enable/disable line
- Monitor and control upstream and downstream traffic, including applying typical access policies, like meters & thresholds
- Read counters



An alternative would be to use FlowObjective also for the BNG-c. However, the current API would require extensions to support matching on the PPPoE header and executing PPPoE specific actions. Moreover, FlowObjective does not allow reading state from the pipeline such as counters. For this reason, we think it's better to define a new API that focuses on BNG-specific capabilities.

Integration with Trellis

We describe here the expected life cycle of a subscriber and the interaction between the different apps in ONOS, from the first packet generated by the RG, to session establishment and termination. We assume the ASG SW is already configured and running, i.e., a suitable netcfg is already provided and the switch has been discovered by ONOS.

We focus on the interaction between the Trellis segment routing app (SR) and BNG-c. To recap, processing of subscriber packets at the ASG SW requires handling of double VLAN tags and the PPPoE header. In this design, SR is responsible for managing pipeline state that deals with

double VLAN termination (push/pop) and routing, while BNG-c is responsible for managing that part of the P4 pipeline that adds/removes the PPPoE header and other BNG-specific capabilities, such as subscriber-level counters. The reason why we choose to split control of the subscriber between SR and BNG-c is simple: SR already provides logic to manage so-called “double-tagged hosts”, independently of PPPoE or other subscriber-specific tunneling protocols, and we prefer to re-use that logic.

When a new subscriber shows up and tries to open a connection from its RG, an initial phase of session establishment is performed. During this phase all packets such as PPPoED, LCP, PAP, CHAP, IPCP, etc., are forwarded as P4Runtime packet-ins to the BNG-c app running on ONOS, which replies by means of P4Runtime packet-outs. If the authentication is successful and the BNG-c deems the session established, the BNG-c app performs two operations: 1) it provisions the necessary pipeline state (i.e, flow rules, groups, etc.) to enable the PPPoE termination, hence rules to remove/add the PPPoE header for upstream/downstream packets, and 2) creates a special “double-tagged host” in the ONOS core which is then used by SR to provision rules that deal with the double VLAN tags and L3 routing between the OLT and the PE Route.

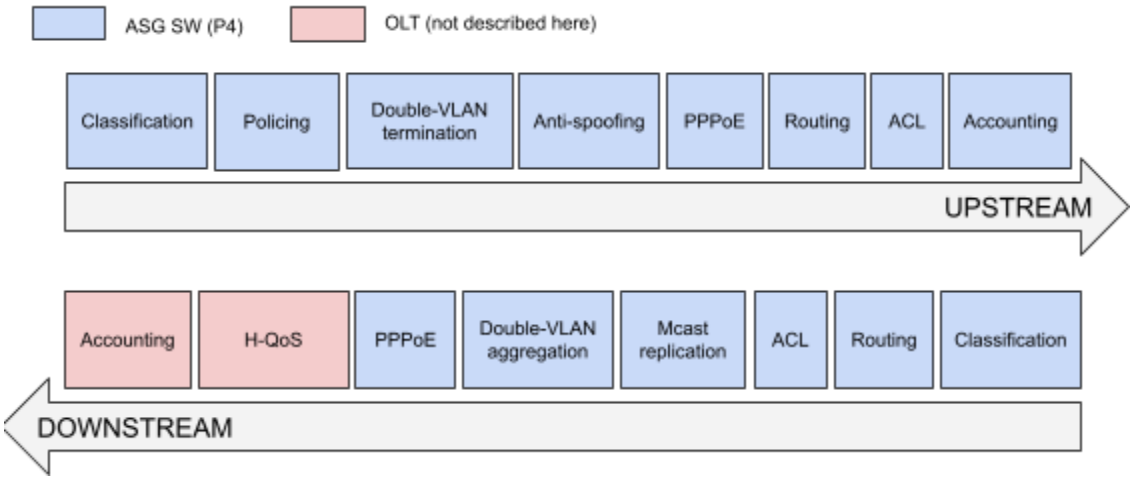
Indeed, each subscriber is represented as a Host in ONOS. The BNG-c app provides HostProvider capabilities, when the negotiation phase is completed, the app creates a new host and submits it to the HostService. The host is provided with its location and annotated with the double VLAN tags (S-tag+C-tag) and IP address. All these information are known by the BNG-c app from the negotiation phase. The SR app listens to events involving double tagged hosts. When a new host (subscriber) is added, the SR app programs the ASG to pop the double VLAN tags and route the packets towards the PE router for the upstream traffic. The host contains all the information needed by the SR app to correctly configure both VLAN tag/untag and routing. SR also programs the ASG to push the double VLAN tags and route the packets towards the correct OLT for the downstream traffic.

When the subscriber disconnects, i.e. when the BNG-c receives the PPP LCP “connection termination” message, it removes the corresponding PPPoE session configuration rules from ASG, and removes the host from the ONOS HostService. SR reacts to the host removal event by removing the pipeline rules regarding double VLAN pop/push and routing of the specific subscriber in the ASG.

P4 pipeline design

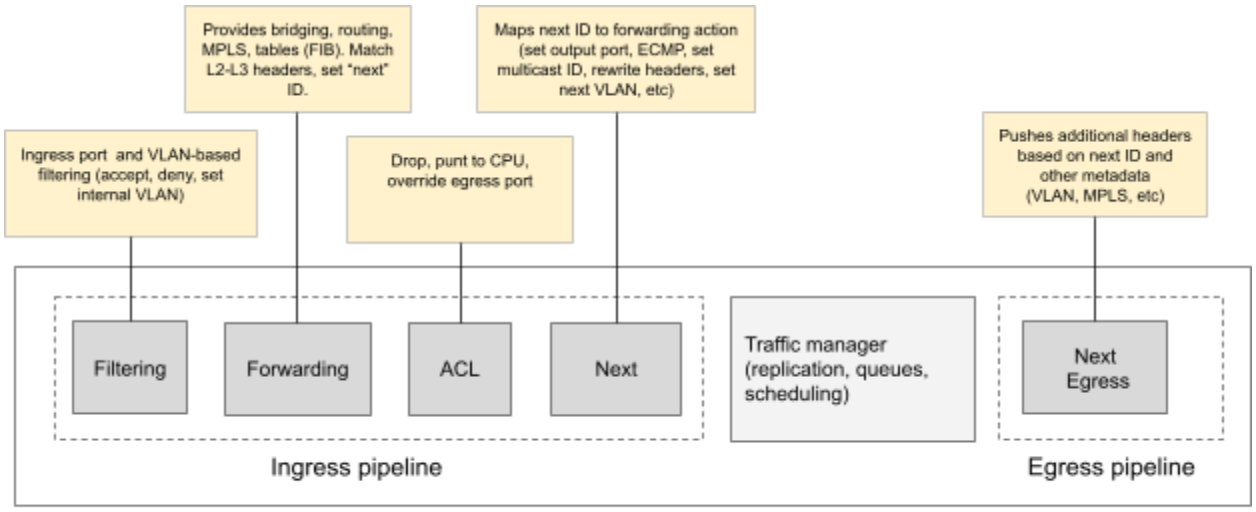
The following picture shows the high-level BNG-u pipeline, with functionalities split between the ASG SW and the OLT. In this section, we discuss only the design of the P4-defined pipeline that implements the BNG-u in the ASG SW. This diagram shows the functional blocks expected to be applied to each packet in both directions, in logical order. When describing the P4 tables used to implement the same blocks, tables might be ordered differently to favor ASIC resource optimization, however, we expect the overall packet processing behavior to be equivalent to this

logical ordering, e.g., accounting should be performed after anti-spoofing and ACL to avoid billing for dropped packets.



Fabric.p4 pipeline recap

As discussed earlier, the BNG-u is implemented as an extension to fabric.p4. The fabric.p4 pipeline is organized as a series of logical stages as depicted in the figure below.



fabric.p4 on V1Model P4 architecture

Each stage provides one or more tables. The function of each stage can be summarized as following:

- **Filtering**: decides whether a packet is to be admitted or not in the pipeline depending on the ingress port and VLAN IDs. If a packet is not admitted, the forwarding and next stages are skipped, and the packet is processed by the ACL table, which might send it to the CPU (packet-in).

- **Forwarding**: this stage provides tables for the different forwarding behaviors, such as bridging, routing (IPv4 and IPv6), MPLS SR, etc. The role of each table is to set a “next ID” (metadata), which will be later used to perform the desired actions, hence introducing a level of indirection.
- **ACL**: it allows to override any previous forwarding decision, as well as cloning or punting packets to the CPU.
- **Next**: this is the last table before packets are sent to queues in the Traffic Manager (TM) and provides tables that match on the next ID and execute actions such as setting the output port, routing, cross-connect, setting the MPLS label, setting the outer VLAN ID, etc. This stage provides handling for double VLAN tags with push/pop and route behaviors, allowing us to re-use this functionality to terminate subscriber tunnels, and instead focus only on PPPoE header processing for the BNG part.
- **Next egress**: this stage is responsible for performing any additional operation associated to the next ID previously set in the ingress pipeline, such as pushing headers like VLAN and MPLS labels. Pushing headers in the egress pipeline allows to save buffering resources in the traffic manager.

Double VLAN handling in fabric.p4

Fabric.p4 supports handlings of double-VLAN tags with behaviors like *pop and route* (upstream) and *route and push* (downstream).
Pop and route is achieved with the following processing:

- **Filtering stage**: it matches on both VLAN headers and sets a packet metadata to signal the egress stage to pop both VLAN headers;
- **Forwarding stage**: the routing table is applied and a next ID set;
- **Next stage**: this stage will perform the actual routing action according to the next ID, rewriting the MAC addresses, and setting the output port of the specific packet.

Route and push, instead, is done with the following processing:

- **Forwarding stage**: the routing table is applied and a next ID set;
- **Next Stage**: it performs the actual routing action, rewriting the MAC addresses, and setting the output port. This stage is also sets both VLAN tags (C-tag and S-tag) as packet metadata to signal the egress pipeline to push the VLANs header to the packet.

BNG-u integration with fabric.p4

Design goals

When integrating the BNG-u functionalities in fabric.p4 we should do so by keeping in mind the following design goals:

1. Keep the pipeline design modular so to facilitate separation of concerns at the control plane, having the Trellis apps control the aforementioned stages/tables, while the BNG-c app should be responsible of controlling just the BNG-related tables.

2. Minimize ASIC resource occupancy, such as number of match-action stages and memory.

The design we present here focuses mostly on the first goal and it's helpful to prototype requirements. We anticipate that additional work will be required to optimize this design for the Tofino ASIC and to support the scale requirements of SEBA, i.e. number of subscribers for a SEBA pod.

Initial P4 implementation and PTF tests

The design described here has been already implemented along with preliminary PTF-based tests that verify the correct processing of packets.

The P4 implementation is available at:

- onos/fabric.p4 with BNG-specific tables defined in bng.p4

The BNG-specific PTF tests are provided as part of fabric-p4test, a suite of data plane tests for fabric.p4. Look for test cases annotated with **`@group("bng")`**:

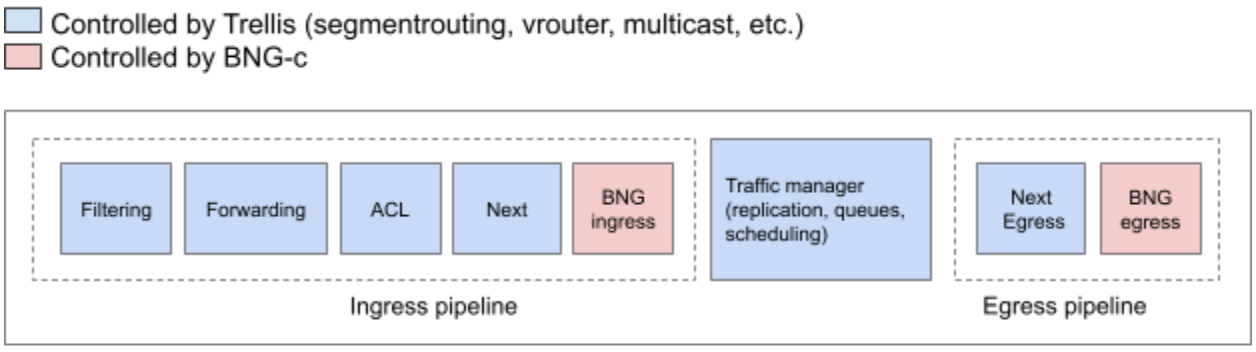
- https://github.com/opennetworkinglab/fabric-p4test/blob/master/tests/ptf/fabric.ptf/test.py#L772

To execute the tests, follow fabric-p4test official instructions. For the impatient (requires Docker):

```
git clone https://github.com/opennetworkinglab/fabric-p4test.git
cd fabric-p4test
./docker_run.sh fabric-bng TEST=bng
```

Overview

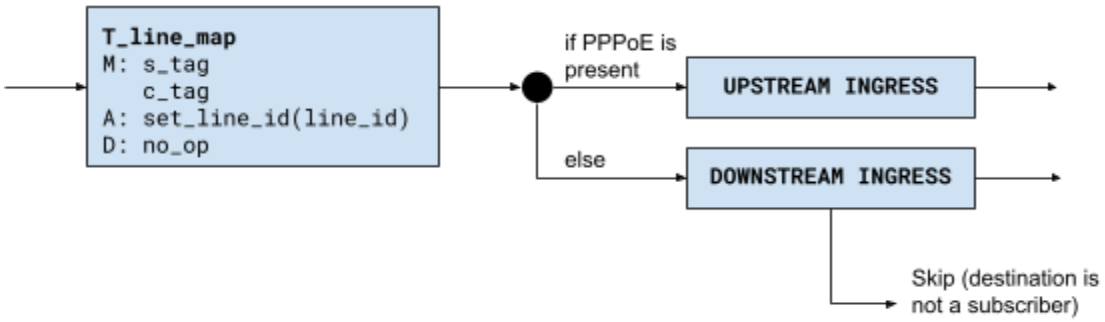
The BNG-u functionalities are achieved by means of two additional stages in the fabric pipeline, BNG ingress and BNG egress.



Fabric.p4 with BNG-u support on V1Model P4 architecture

The BNG ingress stage can be divided in two sub-pipelines: Upstream Ingress and Downstream Ingress. All packets with a valid PPPoE header are deemed BNG upstream packets. If the S_tag and C_tag matches that of subscriber, the line ID is set by `T_line_map` and the packet is processed through the rest of the BNG downstream pipeline, otherwise, the downstream pipeline is skipped and the packet is sent to the Traffic Manager without further modifications.

BNG ingress stage branching

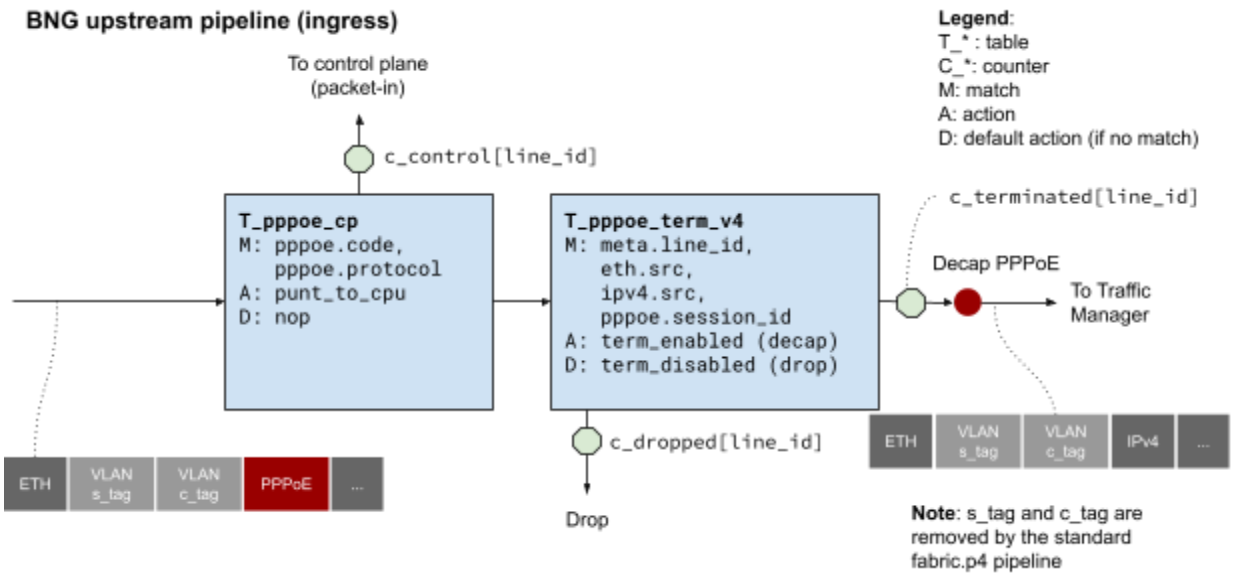


The table **`T_line_map`** provides a mapping between S-tag and C-tag to a line ID (metadata), used to uniquely identify a subscriber in the BNG stages. For the upstream traffic, packets already contains the two VLAN tags in the packet header. For the downstream traffic, the tags are pushed by the Next stage of the fabric.p4 pipeline.

What is a line ID?

Line IDs are used to identify the “access line” from the RG to the BNG (ASG in our case). We assume only one subscriber can be authenticated in a given access line at a given time, as such, **we implicitly use line IDs to uniquely identify subscribers.** Line ID is mapped in the table `T_line_map` from S-tag and C-tag. If we expect multiple subscribers to co-exist on the same access line, i.e, for the same S-tag and C-tag pair, then a refactoring of this pipeline is needed to derive the subscriber ID from other fields, for example the combination of line ID and PPPoE session ID. Defining line_id as a P4 metadata simplifies operations on subscriber-dependent structures such as tables (match on line_id) and counters (indexed by line_id).

Upstream ingress



The upstream ingress pipeline is responsible of:

- For data packets, validate the PPPoE headers and, if the subscriber session has been enabled by the BNG-c, remove PPPoE headers before sending the packet to the fabric.p4 Traffic Manager. Otherwise, if the session is disabled, drop the packet.
- For control plane packets, identified by their PPPoE code and protocol fields, mark the packet to be sent to the BNG-c as a P4Runtime packet-in.

The terminated packets that have already matched on the routing table in the Forwarding stage, will be routed according to its destination IPv4 address, e.g. matching the default route to the Internet. Programming of the Forwarding stage is assumed to be performed by the Trellis app, i.e. defining routes with their next hop (the PE Router in this case), either via static routes or dynamic using the vRouter apps.

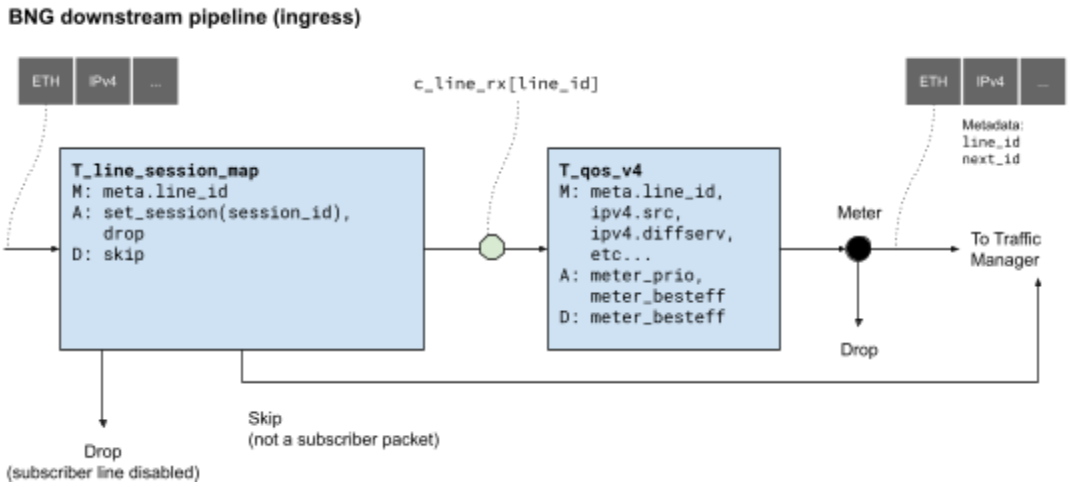
The processing described so far is achieved by 2 tables:

- **`T_pppoe_cp`**: matches on PPPoE control packets to be sent to the BNG-c;
- **`T_pppoe_term_v4`**: terminates data packets, by matching on the line ID, PPPoE session ID, Ethernet and IPv4 source addresses. This table provides anti spoofing capabilities as it guarantees that for a given line ID, only packets coming from the expected subscriber address and PPPoE session ID are terminated, otherwise dropped. This capability is usually defined as Unicast Reverse Path Forwarding (uRPF).

This pipeline provides also 3 counters, each one maintaining a different count value for each line ID:

- **`C_control`**: number of packets sent to the control plane (BNG-c);
- **`C_terminated`**: number of packets terminated (line_enabled);
- **`C_dropped`**: (per line id) number of packets dropped (line disabled or invalid IPv4 source address or PPPoE session ID).

Downstream ingress



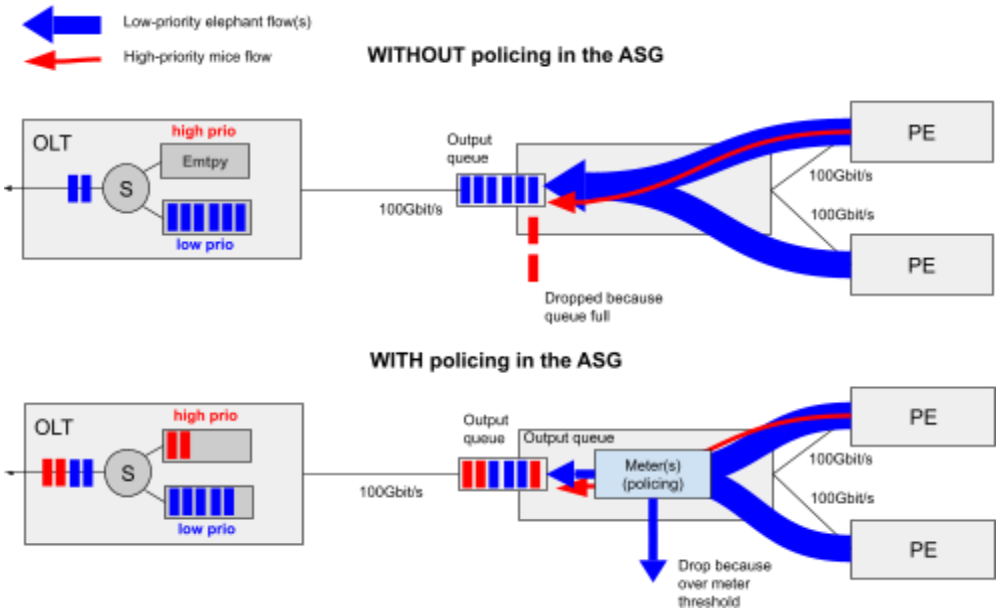
Downstream packets are first processed by **`T_line_session_map`**, a table used to determine the PPPoE session ID from the line ID, and hence the subscriber, to which the packet is destined. This is done by matching on the line ID, expected to be unique for an authenticated subscriber. If the line is enabled, we set the PPPoE session ID as packet metadata, otherwise, the packet is dropped.
Downstream packets have the line ID already set by the initial table (`T_line_map`) of the BNG stage.

Downstream termination of subscriber tunnels requires to push double VLAN tags, the PPPoE header, as well as other routing functions. Since fabric.p4 already support route and push behaviors with double VLAN tags, according to the first design goal, we expect the Trellis apps to be responsible for the provisioning of next IDs and the corresponding “route and push” actions in the Next stage of fabric.p4. In other words, once the packet enters the BNG downstream ingress pipeline, is expected to have both S-tag and C-tag set by the combination of Forwarding and Next stage. Furthermore, destination MAC address is expected to be the subscriber’s and the source MAC address belongs to the ASG (also provided by Trellis).

Since the `T_line_session_map` table matches only line IDs associated with subscribers, a table miss will happen if the packet is not associated with any subscriber. In this case the packet is marked to skip the rest of the BNG pipeline and to be processed by the Traffic Manager. In this way the traffic generated by other services, for example running in the compute nodes, can still be managed in the standard fabric.p4 pipeline.

This pipeline provides also preliminary QoS capabilities such as policing with drop to ensure the OLT is not overloaded. This is achieved by an additional table **`T_qos_v4`** and P4 meters. T_qos_v4 serves as a classifier, matching in a ternary fashion on the line ID, IPv4 source address and diffserv bits, and assigning a class such as best-effort or priority. Depending on the class. The BNG-c assigns each subscriber with a maximum bitrate threshold per class, i.e. configuring the meters, which are then used to perform policing (drop) based on such thresholds.

The idea behind defining multiple traffic class and corresponding meters is to avoid dropping packets that would have been otherwise prioritized by the H-QoS scheduler inside the OLT. The role of the meters inside the ASG SW, is to perform policing to avoid starvation of high-priority flows (e.g., VoIP or IPTV) before they can even reach the H-QoS scheduler. The figure below shows the case where low-priority elephant flows cause high-priority packets to be dropped at the output queue of the ASG. By using policing in the ASG, we can make sure high priority packets can reach the OLT.



Finally, only one counter is defined for the downstream pipeline:

- **`C_line_rx`**: to count subscriber packets received and allowed to walk through the rest of the BNG downstream pipeline.

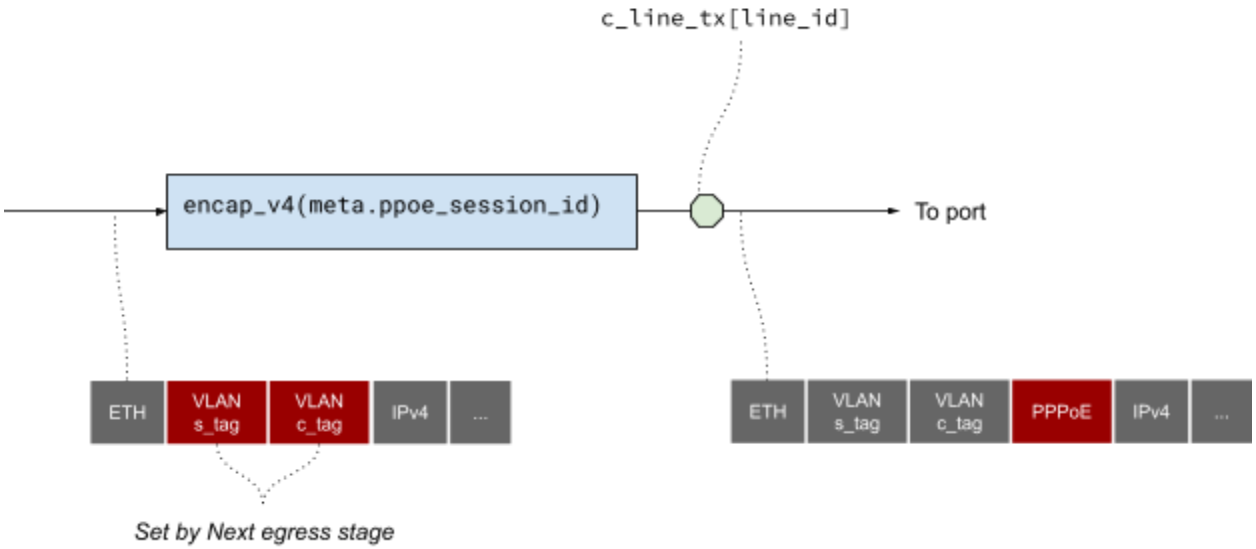
Upstream egress

There's no need of egress processing for upstream traffic.

Downstream egress

This pipeline sits at the egress and it's used to complete processing of downstream packets initiated in the ingress pipeline. Encap_v4 is a P4 action that adds the PPPoE header if the PPPoE session ID has been set in the ingress pipeline. Finally, a counter **`c_line_tx`** is used to count packets that made it through the traffic manager, thus allowing the control plane to be aware of dropped packets because of metering in the ingress pipeline or buffering in the traffic manager.

BNG egress downstream



Missing features

These are features commonly found in other commercial BNG implementations that we did not consider in this design, but we plan to tackle as future work.

IPoE-based service delivery

Wholesale tunnel relay (e.g. via L2TP)

Punt rate limiting

Limit rate of packets punted to the control plane (i.e. BNG-c) with a per-subscriber meter/policer. Control plane packets could be PPPoED, PPP, etc. This is usually referred to as "Excessive Punt Flow Trap" in other BNG implementations.

MTU and fragmentation

Adding VLAN and PPPoE headers, might cause the packet to exceed the minimum MTU on the path to the subscriber, causing it to the discarded. The solution to this problem in a traditional BNG is to perform IPv4 fragmentation. However, fragmentation is a complex process to handle at line rate when using P4 and merchant silicon. As such, we do NOT consider fragmentation for now but we assume all frames received from the PE router to have size compatible with the minimum MTU on the path to the subscriber, even after adding VLAN tags and PPPoE headers.

Moreover, we could add support for Path MTU Discovery (PMTU) by instructing the ASG SW to generate ICMP Fragmentation Needed messages in response to large frames. Some investigation is required, but it's possible that we could implement this process in P4, entirely in the data plane, i.e. generating ICMP Fragmentation Needed messages at line rate, without any involvement of the control plane.

Lawful intercept

Support the ability to instantiate "wiretaps" on a per-subscriber basis as well as sending wiretapped packets to an external collector/mediation device. This can be achieved by means of packet replication groups (multicast/clone) in P4Runtime and might require the ability of encapsulating L2 frames in an additional Ethernet + IP header for delivery to collector. For more information on lawful interception requirements:
https://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r6-1/bng/configuration/guide/b-bng-cg-asr9k-61x/b-bng-cg-asr9k-61x_chapter_0111.html#concept_730F9992F6CA4CB8A044B16A2D8A4F61

Support for L2 and L3 VNF chains on the compute nodes

Support the ability of steering subscriber flows through VNFs in the compute nodes. This can be achieved with L2 "bump-in-the-wire" or L3 services.

Other BNG implementations

Reference:

- https://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r6-1/bng/configuration/guide/b-bng-cg-asr9k-61x/b-bng-cg-asr9k-61x_chapter_0111.html